Interactivity

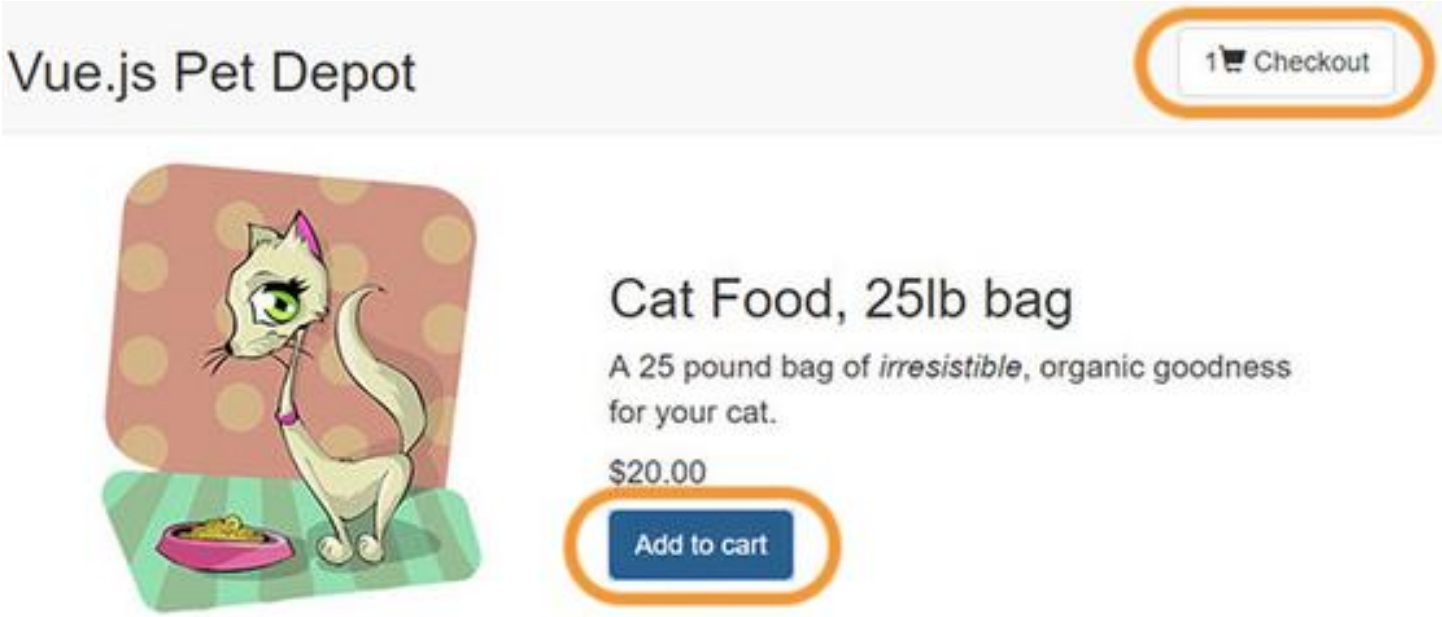
Outline and Learning Objectives

- **Managing Interactions**:
  - to design and develop events management on web-based environments
  - **[Example]** To manage events for **cart interactivity**
- **Managing, Manipulating and Displaying Data** based on **Events and Use Interactions**:
  - to manage data concerning common and frequently developed website functionalities, in terms of:
    - displaying information and manipulating information based on user interactions
  - **[Example]** To **manage the inventory** for a web-based app
- **Managing Interactions within Different Website Areas**:
  - to manage basic **user navigation** within website parts
  - **[Example]** to **manage navigation from products list to the checkout area** for a web-based app
- Suggestions for Reading

Managing Interactions

Final Expected Result for Today (1)

- **We are simplifying: our focus is now mainly on the front-end perspective** (in the second part of the Module we will focus more on the back-end aspects)



- We need to manage interactions for: **inventory, cart, different areas (product list, checkout)**

- **Expected Activities:**

  - **to create a cart**
  - **to display the "Add to cart" Button**
  - **to manage the "on click" event**
    - **to add an item to the cart**
    - to display and update the items count of the cart
  - to create the inventory
  - to limit the possibility to add items to cart only if there are still available items
  - to allow the user to switch from the product list to the checkout areas

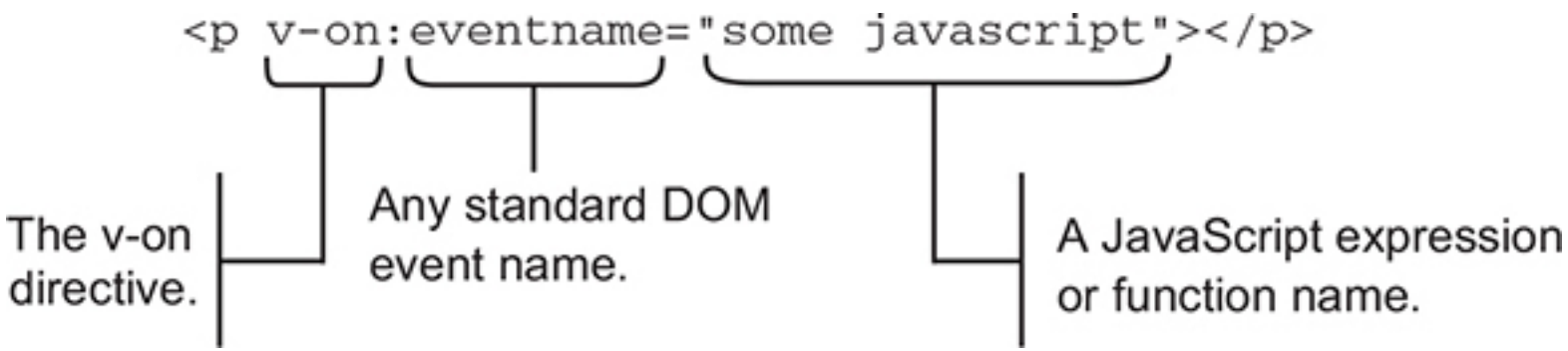
Creating a cart with an Array

What data structure could we use? -> Array (there are more sophisticated solutions; for example, in terms of Object-Oriented Programming, OOP, we could use a Class: Cart)

```
data: {
        sitename: "Vue.js Pet Depot",
        product: { // product information similar to the last time
                id:1001,
                title: "Cat Food, 25lb bag",
                description: "A 25 pound bag of irresistible, organic goodness for your cat.",
                price: 2000,
                image: "assets/images/product-fullsize.png",
        },
        cart: [] // array to store items in shopping cart
},
```

Binding to DOM Events

- Event bindings use the **v-on** directive to bind **a snippet of JavaScript**, or a **function**, to a DOM element
- Any standard DOM events can be indicated (e.g., click, keyup, etc.)




```
<input type="button" value="add" v-on:click='addItem'>
```

```
methods: {
        addItem: function() {
        ...
        }
    }
```

v-on shorthand @

Instead of using **v-on** , you can replace it with the @ symbol

```
<input type="button" value="add" v-on:click='addItem'>
```

Accordingly, the next one is equivalent to the previous one

```
<input type="button" value="add" @click='addItem'>
```

Bind an “event” to the “Add to Cart” Button

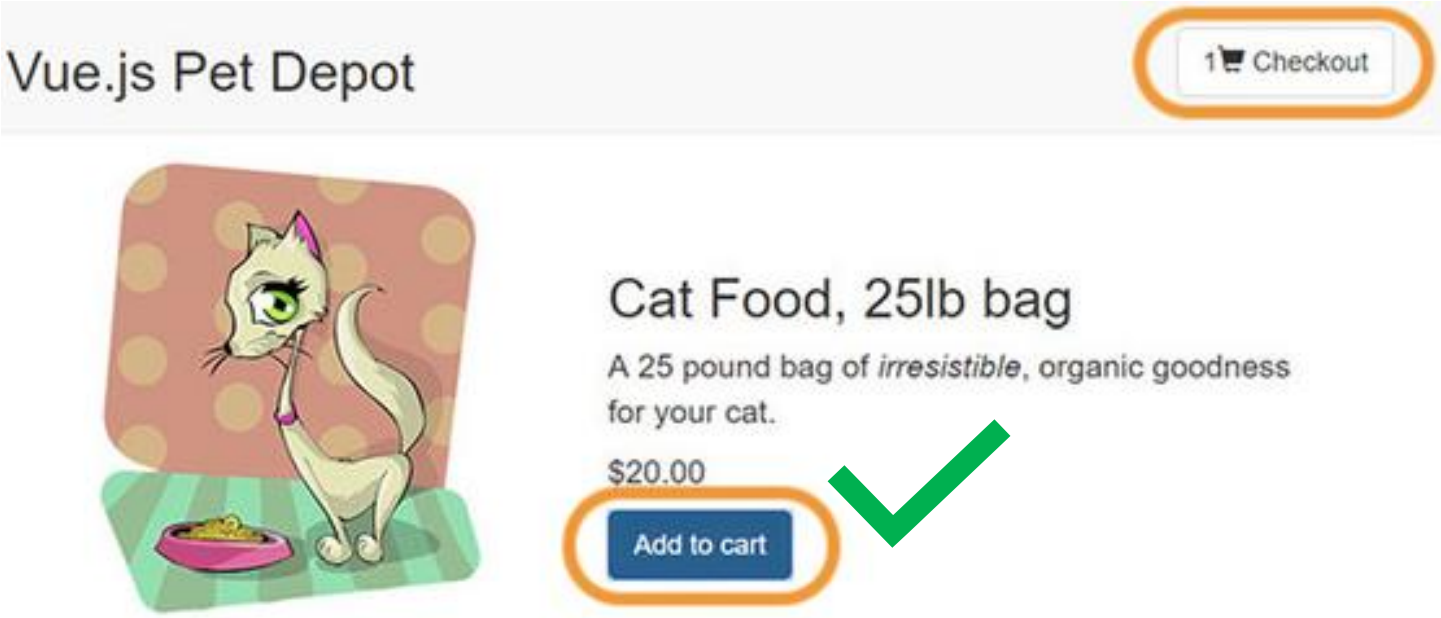
First, we create the event function

```
methods: {
        addToCart: function() {
                this.cart.push( this.product.id );
        }
}
```

- **(our solution)** Adding a product to the cart means pushing the product’s **id** property from the product data onto the **cart** array
- **(another solution)** While pushing the product into the cart array would push a reference to the product object defined in our data, not a copy

The “Add to cart” Button and Event

```
<button v-on:click="addToCart">
        Add to cart
</button>
```



- to create a cart
- to display the “Add to cart” Button
- to manage the “on click” event
    - to add an item to the cart

We still need to cover the other interactions and related management

Current Solution: HTML

- Combining all the elements covered so far

```
<div id="app">
        <header>
                <h1 v-text="sitename"></h1>
        </header>
        <main>
                <figure>
                        <img v-bind:src="product.image">
                </figure>
                <h2 v-text="product.title"></h2>
                <p v-html="product.description"></p>
                <p>Price: {{product.price}}</p>
                <button v-on:click="addToCart">
                        Add to cart
                </button>
        </main>
</div>
```

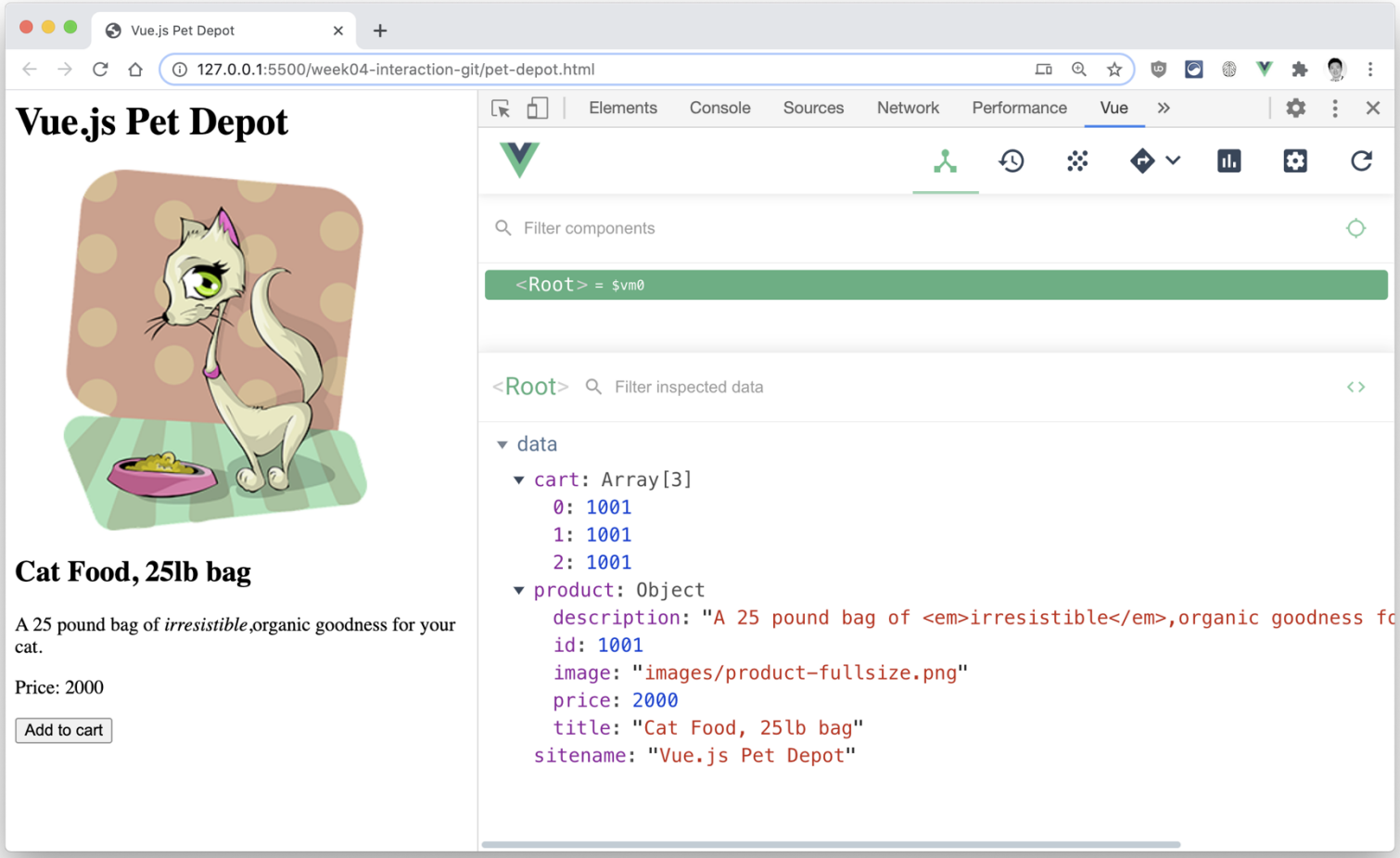
Current Solution: Vue.js

- Combining all the elements covered so far

```
var webstore = new Vue({
        el: '#app’,
        data: {
                sitename: 'Vue.js Pet Depot’,
                product: {
                        id: 1001,
                        title: "Cat Food, 25lb bag",
                        description: "A 25 pound bag of
                        irresistible organic goodness for
                        your cat.",
                        price: 2000,
                        image: "images/product-fullsize.png"
                },
                cart: []
        },
        methods: {
                addToCart: function () {
                        this.cart.push(this.product.id);
                }
        }
});
```

Inspecting Our Solution with vue-devtools

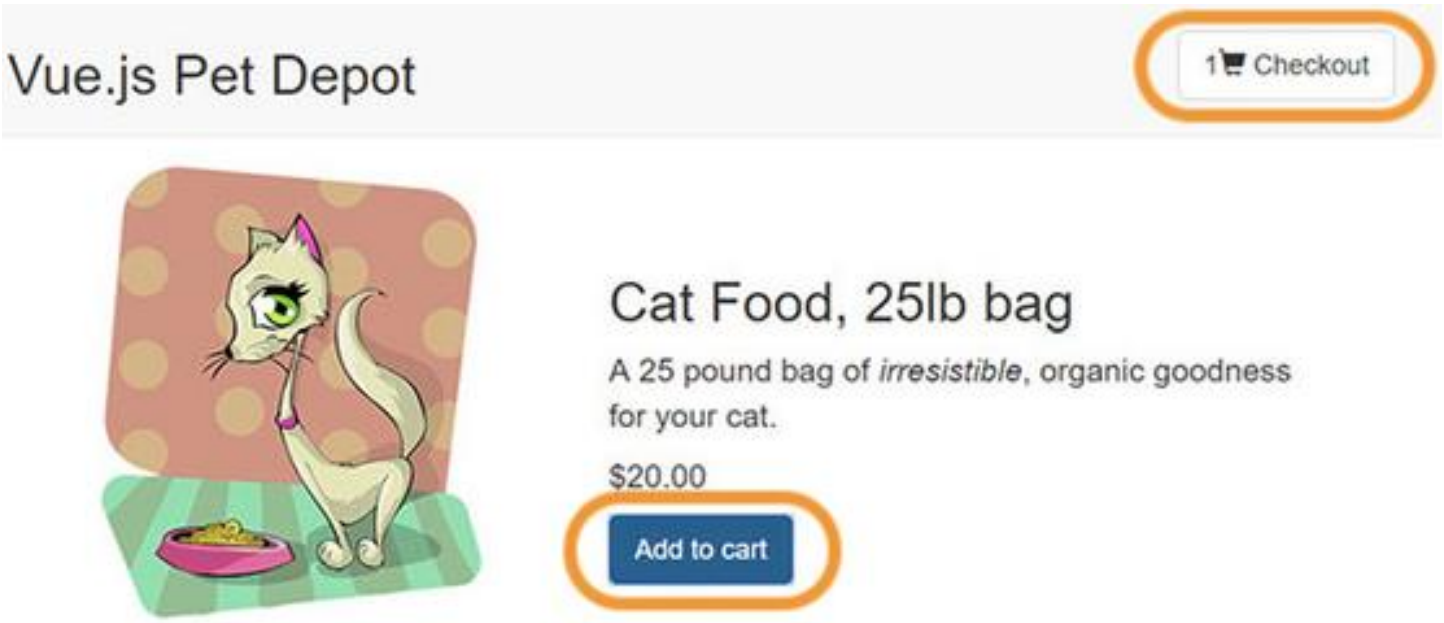
- On your App, right click -> “**Inspect**”
- Click on the right tab called “**Vue**”
- Click on the ”**<Root>**” element to start the inspection
- Verify in **data** that the **id** of the **product** is added to the **array** after clicking on the “**Add to cart**” button
- **IMPORTANT**: sometimes, to see the most updated results, it is required to refresh **vue-devtools** by clicking again on the ”**<Root>**” element, or on the **update icon** (on the **top right**)



Managing, Manipulating and Displaying Data based on Events and User Interactions

Final Expected Result for Today (2)

- **We are simplifying: our focus is now mainly on the front-end perspective** (in the second part of the Module we will focus more on the back-end aspects)



- We need to manage interactions for: **inventory, cart, different areas (product list, checkout)**

- **Expected Activities:**
  - to create a cart
  - to display the “Add to cart” Button
  - to manage the “on click” event
    - to add an item to the cart
    - **to display and update the items count of the cart**
  - **to create the inventory**
  - **to limit the possibility to add items to cart only if there are still available items**
  - to allow the user to switch from the product list to the checkout areas

Computed Properties

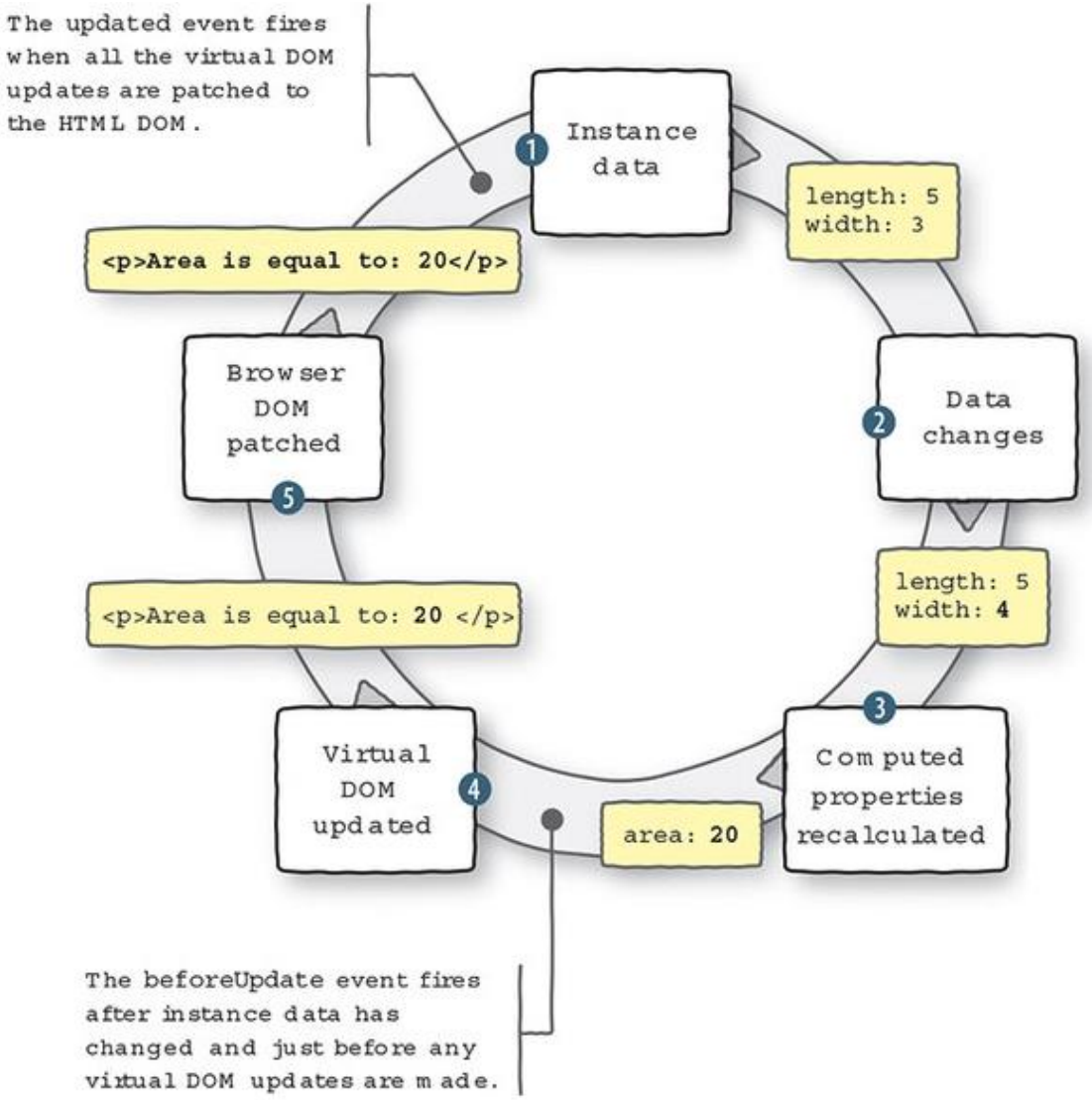
- [**Data Properties**] most of the properties in the **data** could be retrieved from an external source (e.g., **Database**, **Server**, **Middleware**)
- [**Computed Properties**] dynamic elements prepared and managed at the **view-model** level and used mainly at the **view** level

- [**Example1: Chain 1**]
  - **result shown**: "Summer Winters" ;
  - **computed**: fullName -> firstName + lastName
- [**Example2: Chain 2**]
  - **result shown**: "Ms Summer Winters" ;
  - **computed**: fullNameWithPrefix -> prefix + fullName -> firstName + lastName

Computed Properties and Vue Event Management

Example for the calculation of the area of a rectangle: when **width** or **length** are **updated**, also the **computed property** and the **elements displayed** are **updated**

```
<div id="app">
  <p>Area is equal to: {{ area }}</p>
  <p>
    <button v-on:click="length += 1">Add length</button>
    <button v-on:click="width += 1">Add width</button>
  </p>
</div>
<script type="text/javascript">
  var app = new Vue({
          el: '#app',
          data: {
                    length: 5,
                    width: 3
                  },
                  computed: {
                    area: function() {
                      return this.width * this.length;
                  }
              }
...
```



Computed Properties vs Methods

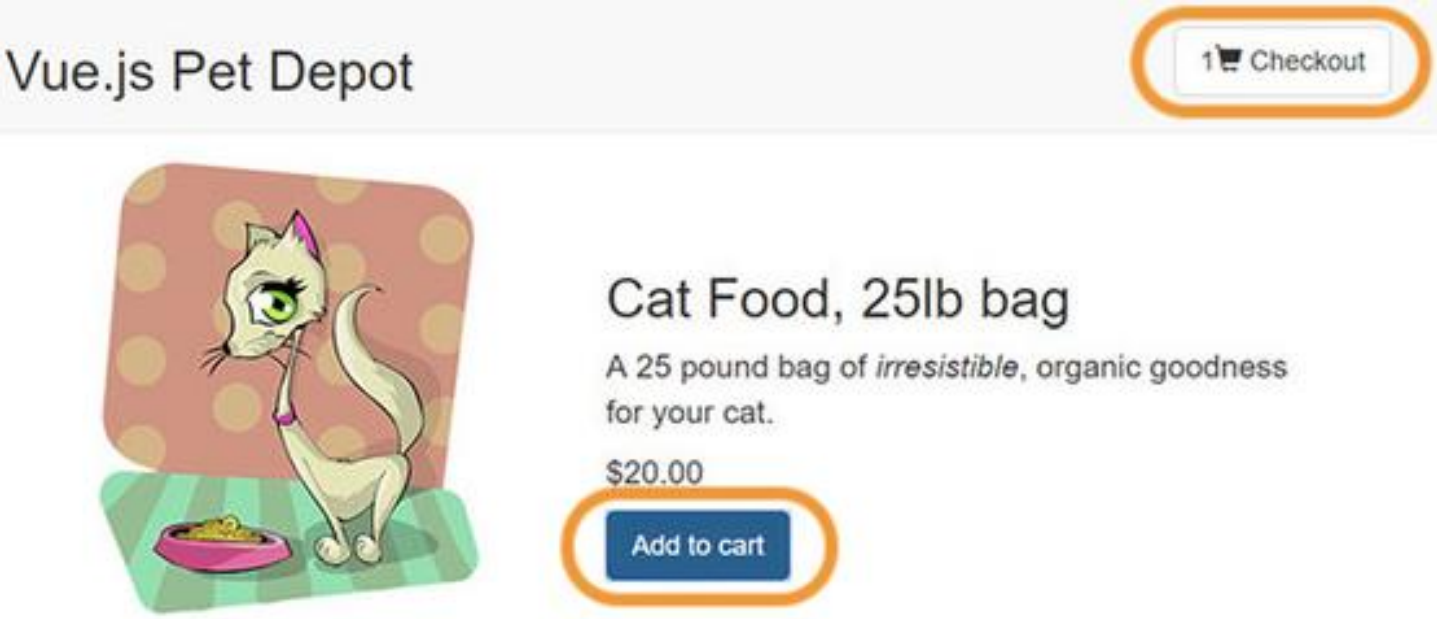
- [**Methods**] executed when called (used similarly to Javascript functions):
  - **accept parameters**
- [**Computed Properties**] recomputed automatically when other inner (inside of its function) **data** variables are updated:
  - very useful for designing and developing robust **user interactions**, because also all the elements binded in the **view will be updated** automatically (**2-way binding**)
  - **DO NOT accept parameters**

```
<script type="text/javascript">
  var app = new Vue({
          el: '#app',
          data: {
                  length: 5,
                  width: 3
                },
                computed: {
                  area: function() {
                    return this.width * this.length;
                }
            }
…
```

Computed Properties and Our App

- The **checkout button** will show the **number of items in the cart**
- We will use **computed property** to achieve this
- Computed properties **can be bound to the DOM element** like any other property defined in the **data**
- Its value is usually derived from the current state of an application

- The **cartItemCount should not be in the data object**
  - because its value changes as the result of user interaction




```
computed: { // the Computed Property object
        cartItemCount: function() { // the property name
                // its value is calculated when it is called
                return this.cart.length || "";
        }
},
```

Checkout Button

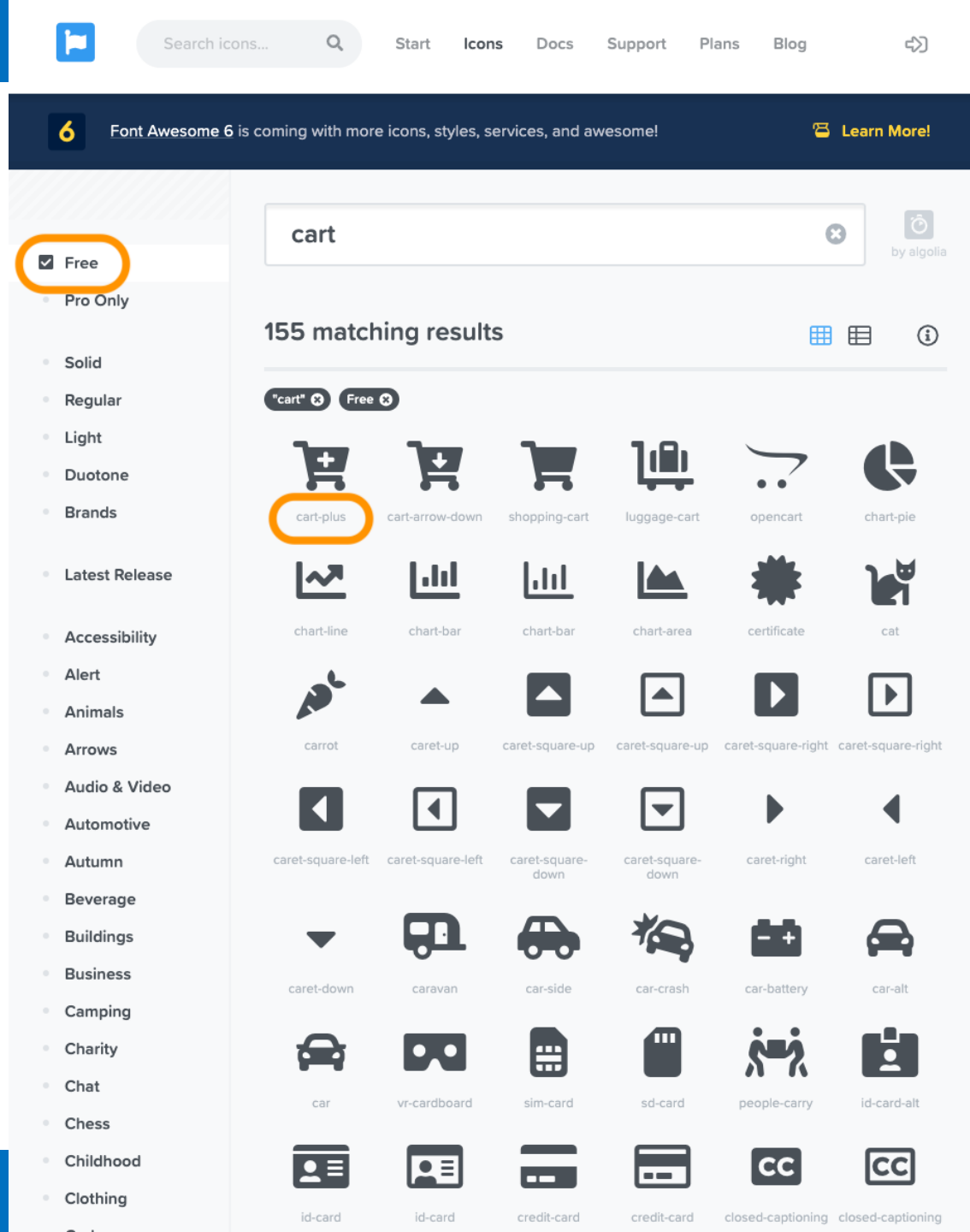
Checkout Button and Font Awesome

- How to show the Checkout Button with an Icon within the text?
- To show it by using a special character (representing an icon) with Font Awesome


```
<head>
        // We are using Font Awesome to create the cart icon
        <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.1/css/all.min.css">
</head>
<body>
        <header>
                <h1>{{ sitename }}</h1>
                <button>
                        <!-- 'cartItemCount' is used the same way as a data property. -->
                        {{ cartItemCount }}
                        <!-- add the cart icon -->
                        <span class="fas fa-cart-plus"></span> Checkout
                </button>
        </header>
...
</body>
```

Font Awesome

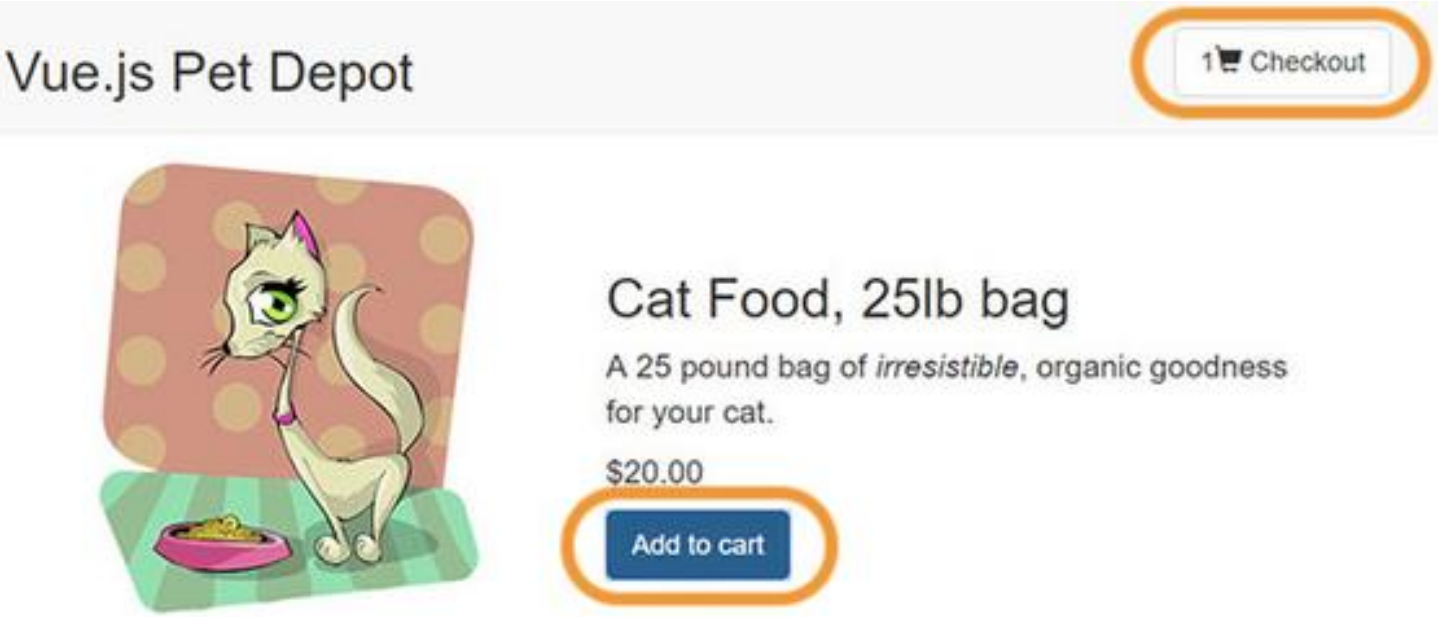
- It allows you to add icon to your page as text instead of image; you need to first load it as an external css file
- **<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/fontawesome/5.15.1/css/all.min.css">**
- Then search for the icon you want, such as 'cart' at Font Awesome
- Make sure to choose the **'free'** icons
- Note the code for your icon, In this case '**cart-plus**'
- Add the icon to your page like:
    - ****
- The class **fas** means it is a Font Awesome icon
- The other class selects the icon (**fa-** in front of it**)**:
    - **fa-icon-code**



Managing Inventory Interactions

Inventory

- We need to make sure that when the "**Add to cart**" button is clicked we have enough **products** in the inventory -> we need an **inventory**
- We will us a new property **availableInventory** to record the stock level




```
data: {
        sitename: "Vue.js Pet Depot",
        product: {
                id: ... ,
                title: ... ,
                description: ... ,
                price: ... ,
                image: ... ,
                availableInventory: 5
        }
        cart: []
}
```

Strategy for Checking the Stock Level

- We do not want to change the **availableInventory**
  - because that should only happen after user finished checkout
- But we do want to restrict the amount of product a customer can add to their cart
  - It should not be more than the **availableInventory**

Inventory and Strategy for Checking the Stock

Strategy for Checking the Stock Level

- We do not want to change the **availableInventory**
  - because that should only happen after user finished checkout
- But we do want to restrict the amount of product a customer can add to their cart
  - It should not be more than the **availableInventory**
- We use a computed property **canAddToCart** to check this
- Note that we use **the other computed property cartItemCount** inside the new computed property **canAddToCart**

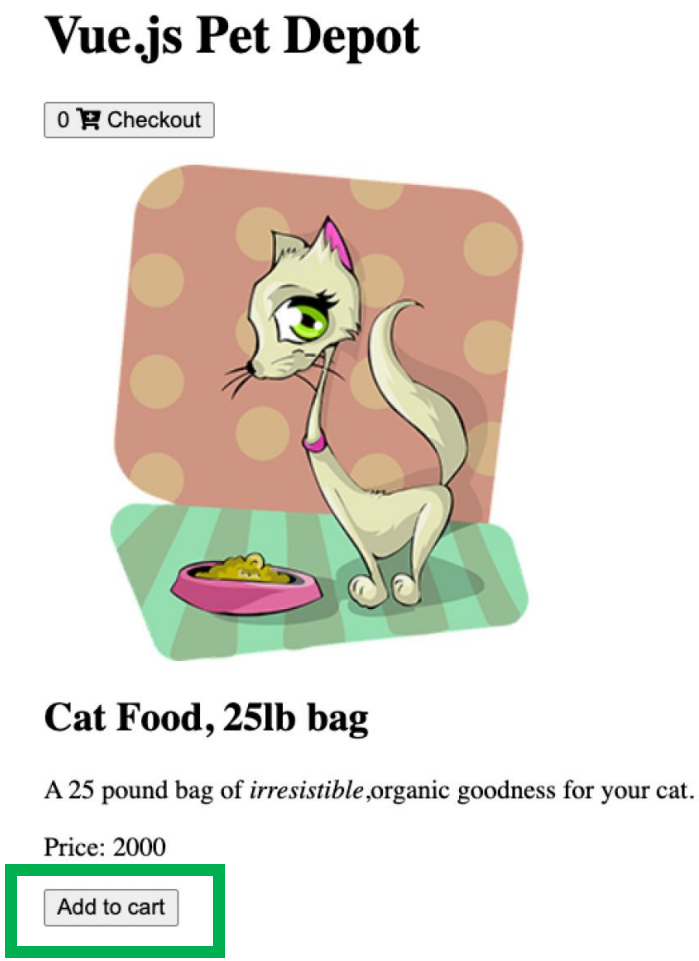
```
computed: {
        cartItemCount: function() {
                ...
        },
        canAddToCart: function() {
                return this.product.availableInventory > this.cartItemCount;
        }
}
```

Managing Button Interactions

2 Possible Strategies

When all the available items of a **product** have been added to the **cart**, the **“Add to cart” button** can be either:

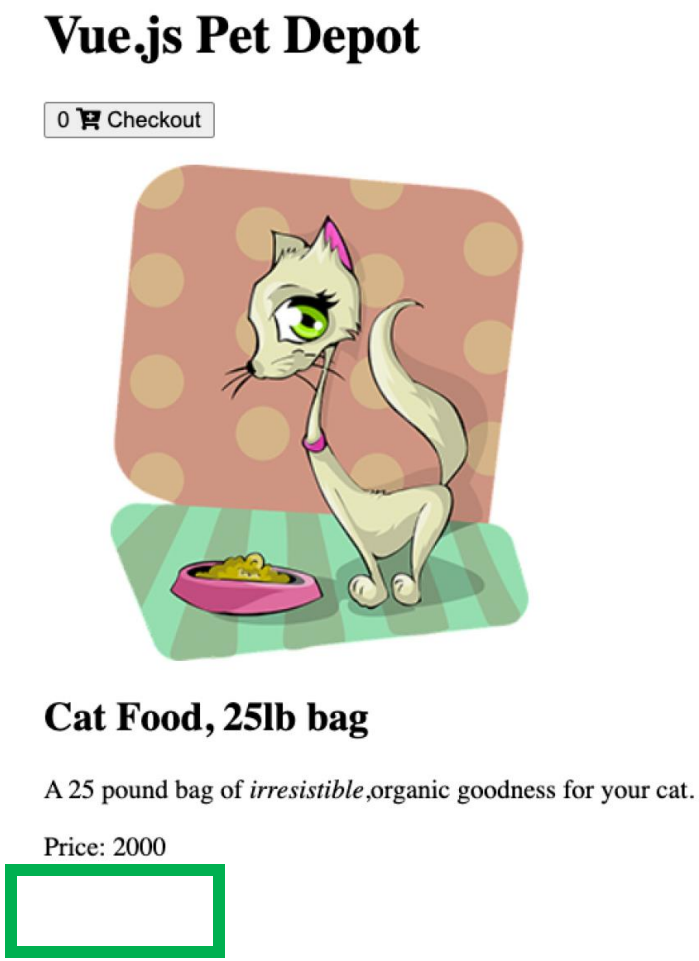
- hidden
- disabled



Hide the Button with v-show

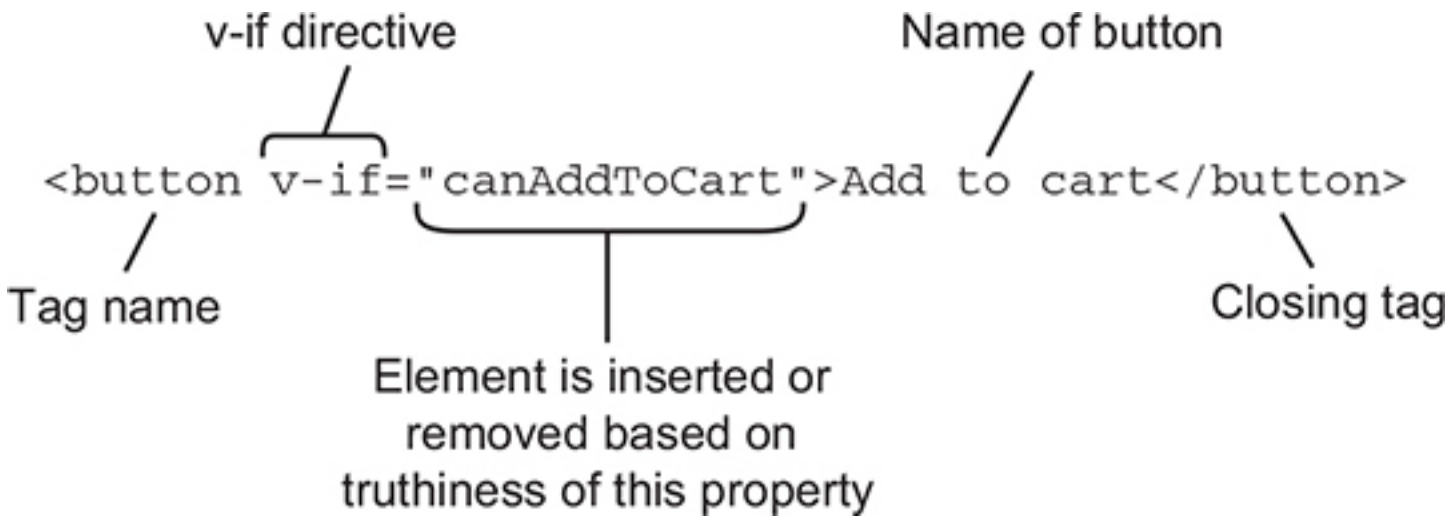
- We will stop a customer from adding more products if the number in the cart is more than the stock level
- We do this by hiding the **'Add to cart'** button
- We can use the **v-show** for this.
- It only shows a HTML element if the condition is **true**
- If the condition is **false**, Vue sets the element's **display CCS property** to **none as an inline style**. This hides the element, but it is **still present in the DOM**

```
<button v-on:click="addToCart" v-show="canAddToCart">
        <!-- only show the button when 'canAddToCart' is true -->
        Add to cart
</button>
```



A Better Solution: a Disabled Button

- It is rare for an e-commerce site to actually **make a button disappear**
- Which is **not really a good user experience**
- Instead, **more likely a button is disabled** to achieve a similar effect
- This can be done with **v-if** and **v-else**
- If the **canAddToCart** is **true** the button appears, if not, the button does not appear -> **Vue removes from the DOM the part related to the false condition**



Disable the Button with v-if and v-else

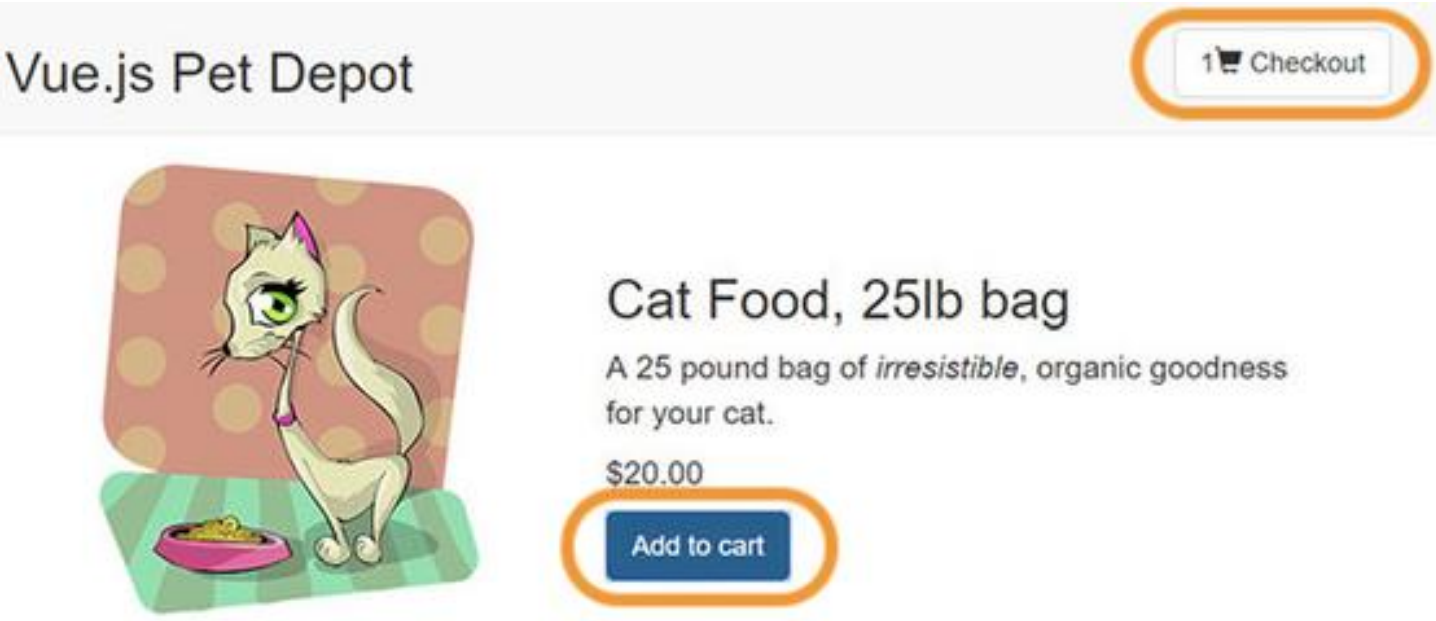
- **[The idea]** we will have **2 similar buttons**: one is enabled, and the other is disabled
- We show the enabled button when user can add more products, and the other button when the user cannot
- We use **v-if** and **v-else** to select which button is shown

```
<!-- This button will be displayed when 'canAddToCart' is True -->
<button v-on:click="addToCart" v-if="canAddToCart">
        Add to cart
</button>
<!-- This button will be displayed otherwise -->
<button disabled='disabled' v-else>
        Add to cart
</button>
```

Managing Interactions within Different Website Areas

Final Expected Result for Today (3)

- **We are simplifying: our focus is now mainly on the front-end perspective** (in the second part of the Module we will focus more on the back-end aspects)



- We need to manage interactions for: **inventory, cart, different areas (product list, checkout)**

- **Expected Activities:**
  - to create a cart
  - to display the “Add to cart” Button
  - to manage the “on click” event
    - to add an item to the cart
    - to display and update the items count of the cart
  - to create the inventory
  - to limit the possibility to add items to cart only if there are still available items
  - **to allow the user to switch from the product list to the checkout areas**

Toggling the Checkout Page

- **So far there is no checkout page yet**
- We will add a **checkout page** that **becomes visible** when the **checkout button** is clicked:
  - **a first click** of the **checkout button** will show the **checkout page**
  - **a second click** of the **checkout button** will hide the **check out page**, i.e. show the **product page** again

```
data: {
   showProduct: true,
   ...
},
methods: {
   ...
   showCheckout() {
         this.showProduct = this.showProduct ? false : true;
   },
}
```

- The **Javascript Ternary Operation** above is equivalent to:

```
if (this.showProduct) this.showProduct = false;
else this.showProduct = true;
```

Display the Checkout Page

- Now we will use the **checkout button** to change the value of **showProduct**

```
<button v-on:click='showCheckout’>
        {{cartItemCount}}
        <span class="fas fa-cart-plus"></span>
        Checkout
</button>
```

- We will use **v-if** to toggle the checkout page based on the value of **showProduct**

```
<main>
        <div v-if='showProduct’>
                <!-- the code for the product page -->
                ...
        </div>
        <div v-else>
                <!-- the code for the checkout page -->
                ...
        </div>
</main>
```

- Clicking the **Checkout button** will toggle between the **product page** and the empty **checkout page**

Suggestions for Reading

Reading

Chapter 3 of the **“Vue.js in Action”** textbook

Questions?